From Details to Done

A Test-Driven Approach to Software Development

Steve Jewett

Systems & Software Technology Conference 2011

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAY 2011		**00-00-2011 to 00-00-2011**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
From Details to Done. A Test-Driven Approach to Software Development	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Boeing Defense, Space & Security,PO Box 516,St. Louis,MO,63166**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Presented at the 23rd Systems and Software Technology Conference (SSTC), 16-19 May 2011, Salt Lake City, UT. Sponsored in part by the USAF. U.S. Government or Federal Rights License

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**25**	

Topics

- **Moving Tests Forward**
- **3 Rules of Test-Driven Development (TDD)**
- **TDD in Unit, Integration and Acceptance Testing**
- **Comprehensive TDD Process**
- **Pros and Cons of TDD**
- **Q&A**

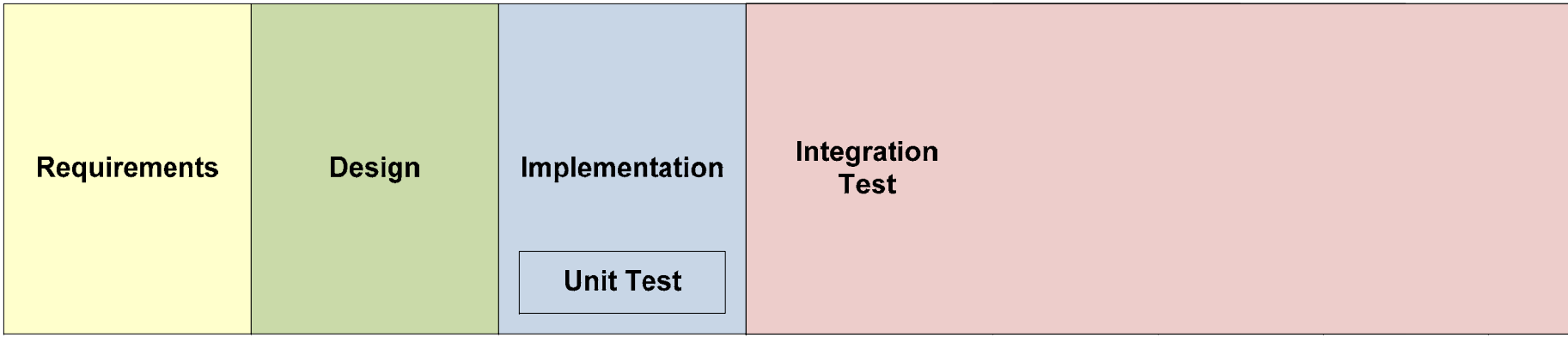
Traditional Development Cycle

Testing Follows Implementation:

Unit tests are executed after modules are completed.

Integration testing follows implementation.

Acceptance testing begins at the end of integration.



Moving Tests Forward

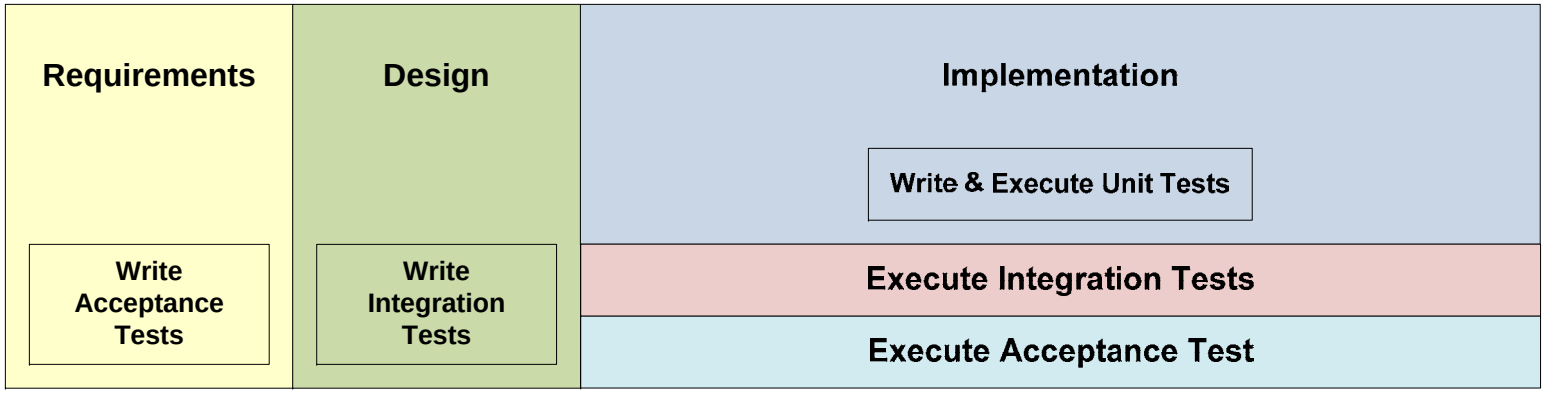
Testing Occurs Before Implementation:

Acceptance tests are developed as part of the requirements.

Integration tests are developed as part of the design.

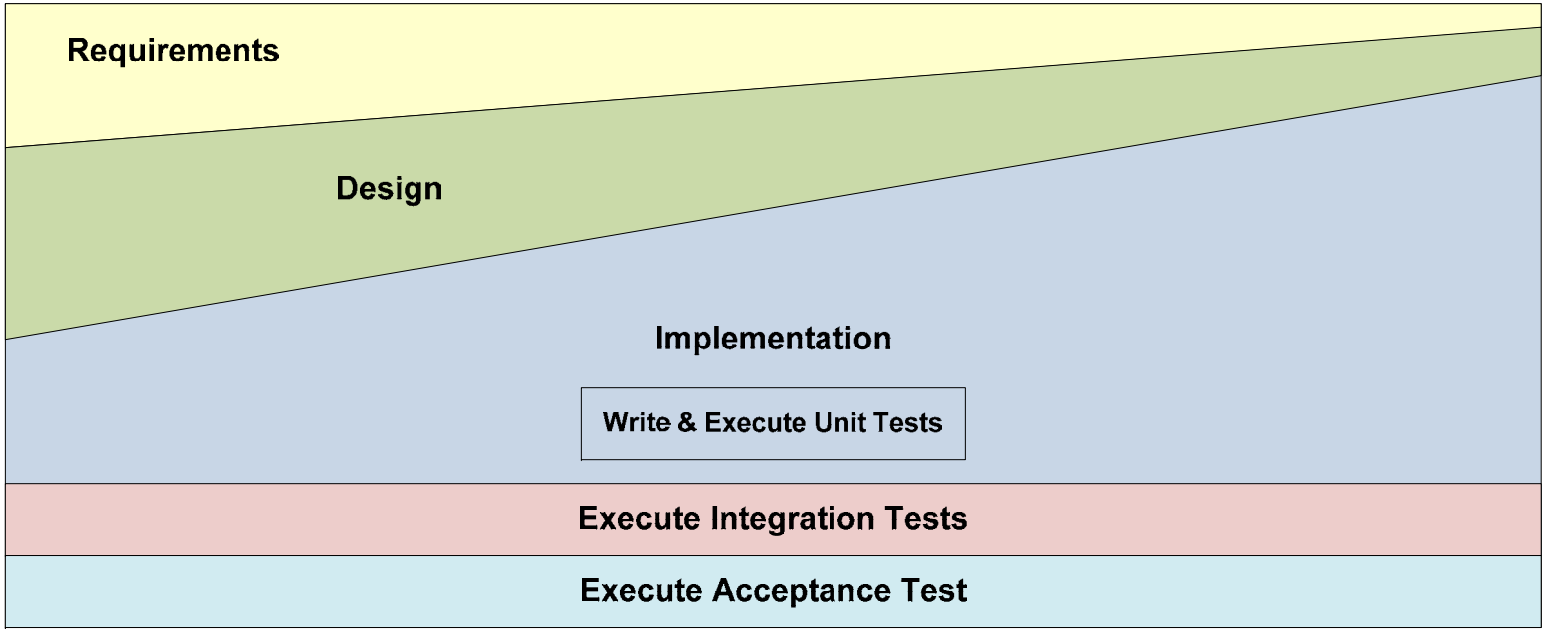
Unit tests are developed as part of the implementation.

Test are executed throughout implementation; test failures drive what to do next.



Testing in an Agile Development Cycle

Agile Development is not phase-oriented, so tests are executed throughout the cycle, not just during implementation.



How Does Testing Drive Development?

Test-Driven Development (TDD) says to create tests first and let them drive implementation. The three rules of TDD demonstrate how to do that.

3 Rules of TDD

1. ~~Write production code~~ (Do work) only to pass a failing test.
2. ~~Write~~ (Do) only enough ~~test code~~ (testing) to fail.
3. ~~Write~~ (Do) only enough ~~production code~~ (work) to pass.

TDD at the Unit Test Level

Unit tests are created by developers to add functionality to a class or module.

At the unit test level the three rules are manifest in the "red-green-refactor" approach:

Red-Green-Refactor

Write a unit test that fails.

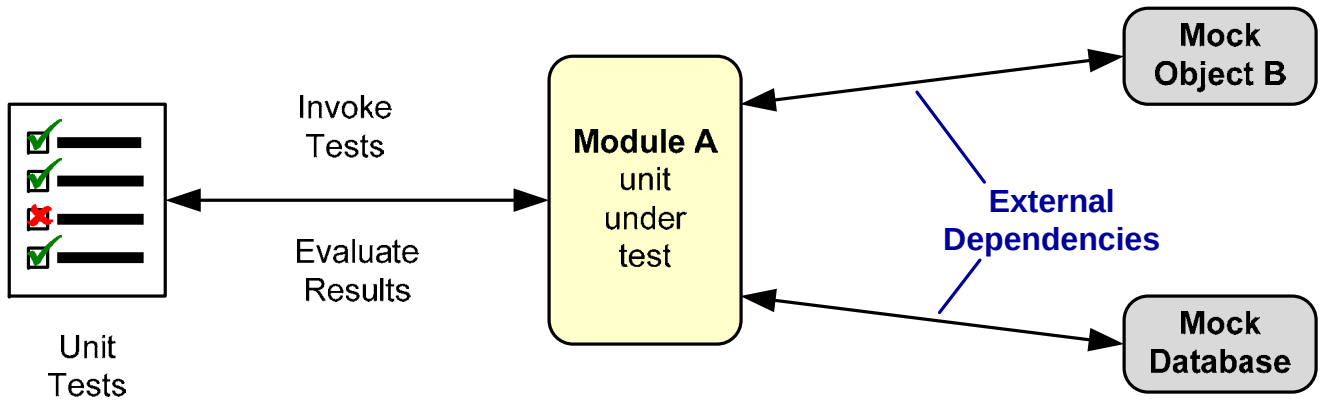
Write production code to make the test pass.

Clean up both test and production code.

for example …

TDD at the Unit Test Level (cont'd)

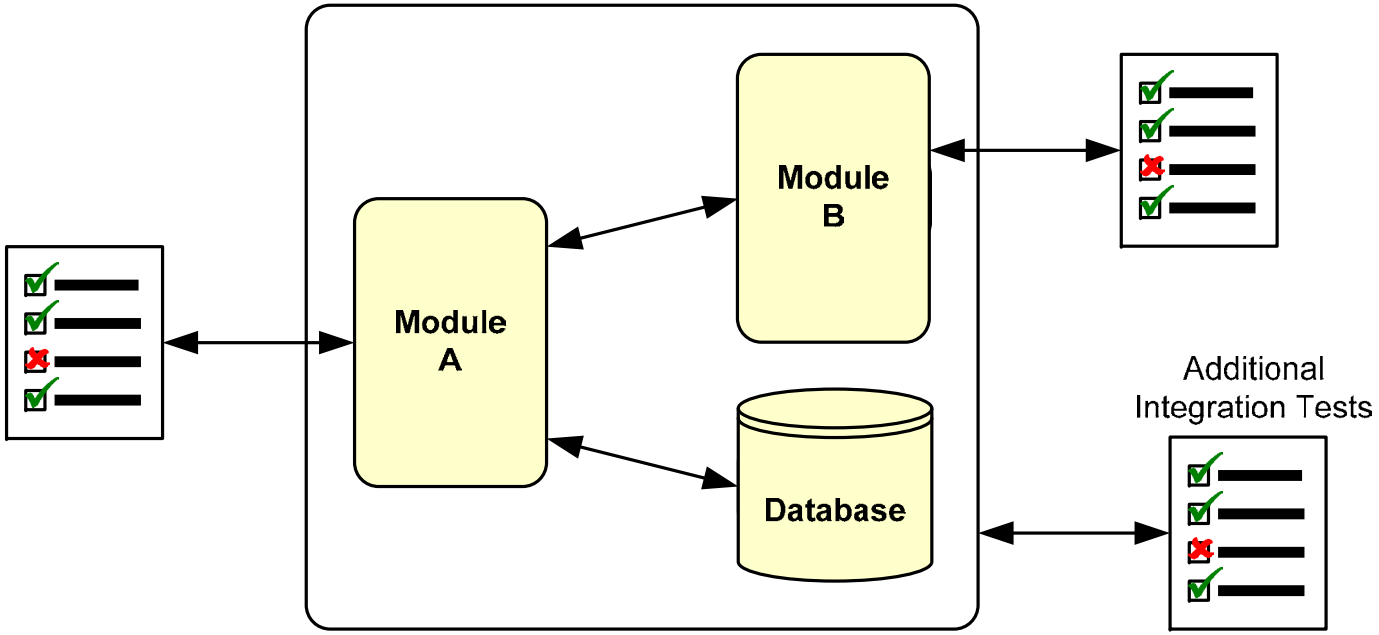
Using the Red-Green-Refactor approach, developers create unit tests for individual modules as they add functionality.



External dependencies are handled by creating mock objects.

TDD at the Integration Test Level

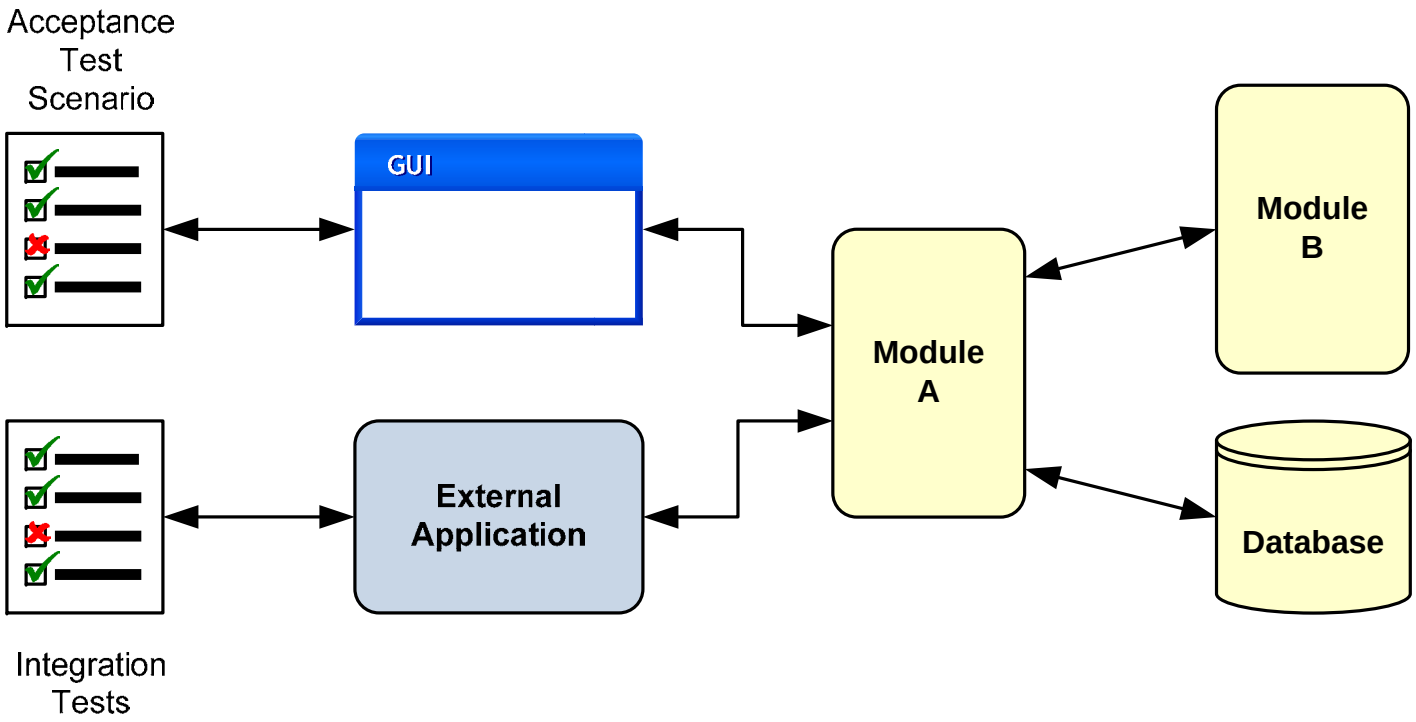
Initial integration tests are the unit tests with real components replacing mock objects.



Additional integration tests may be needed to address scaling, loading or speed.

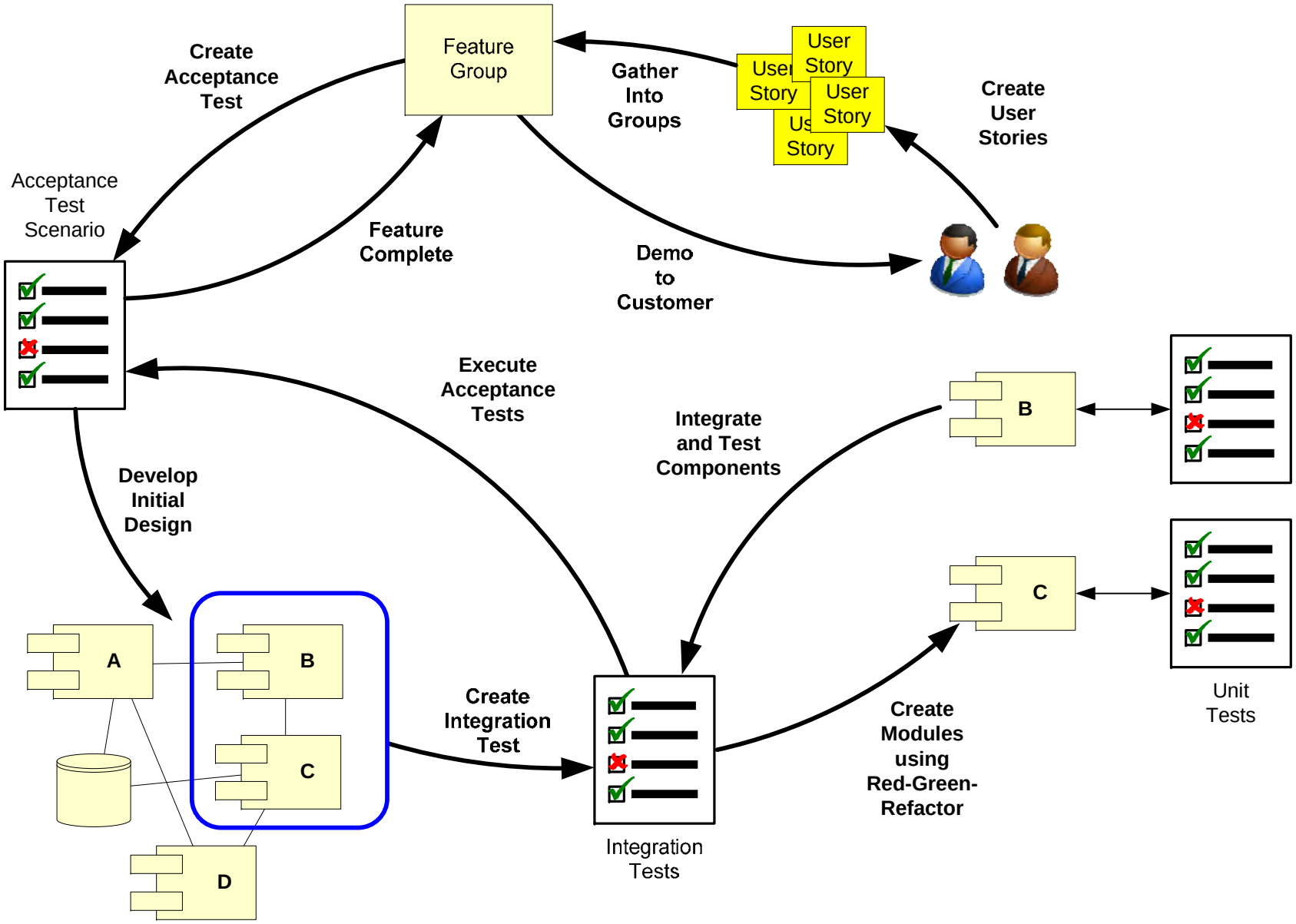
TDD at the Acceptance Test Level

Acceptance tests may take the form of use case scenarios executed via a user interface …



… or they may be the integration tests from an external application.

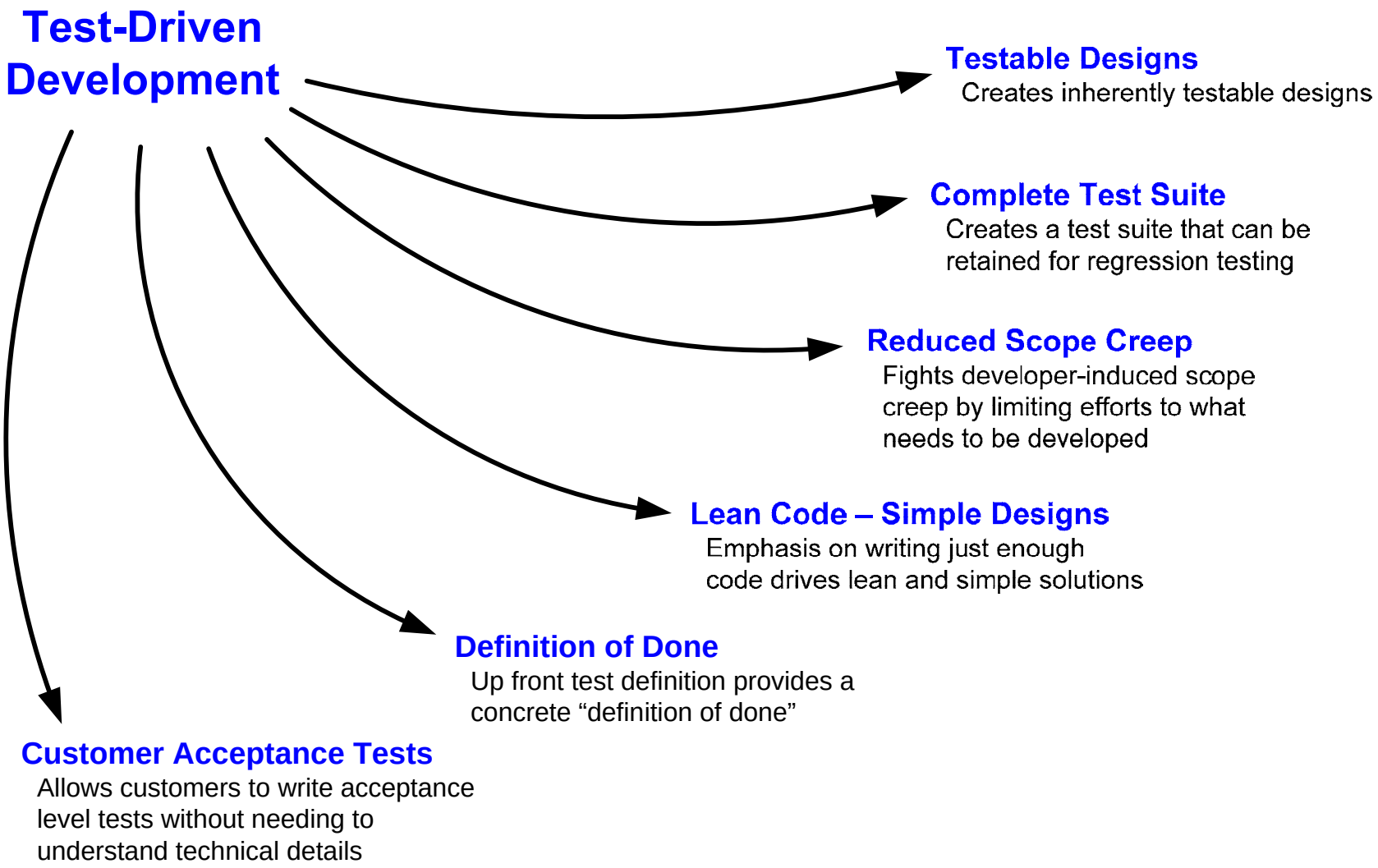
Driving Development with Tests



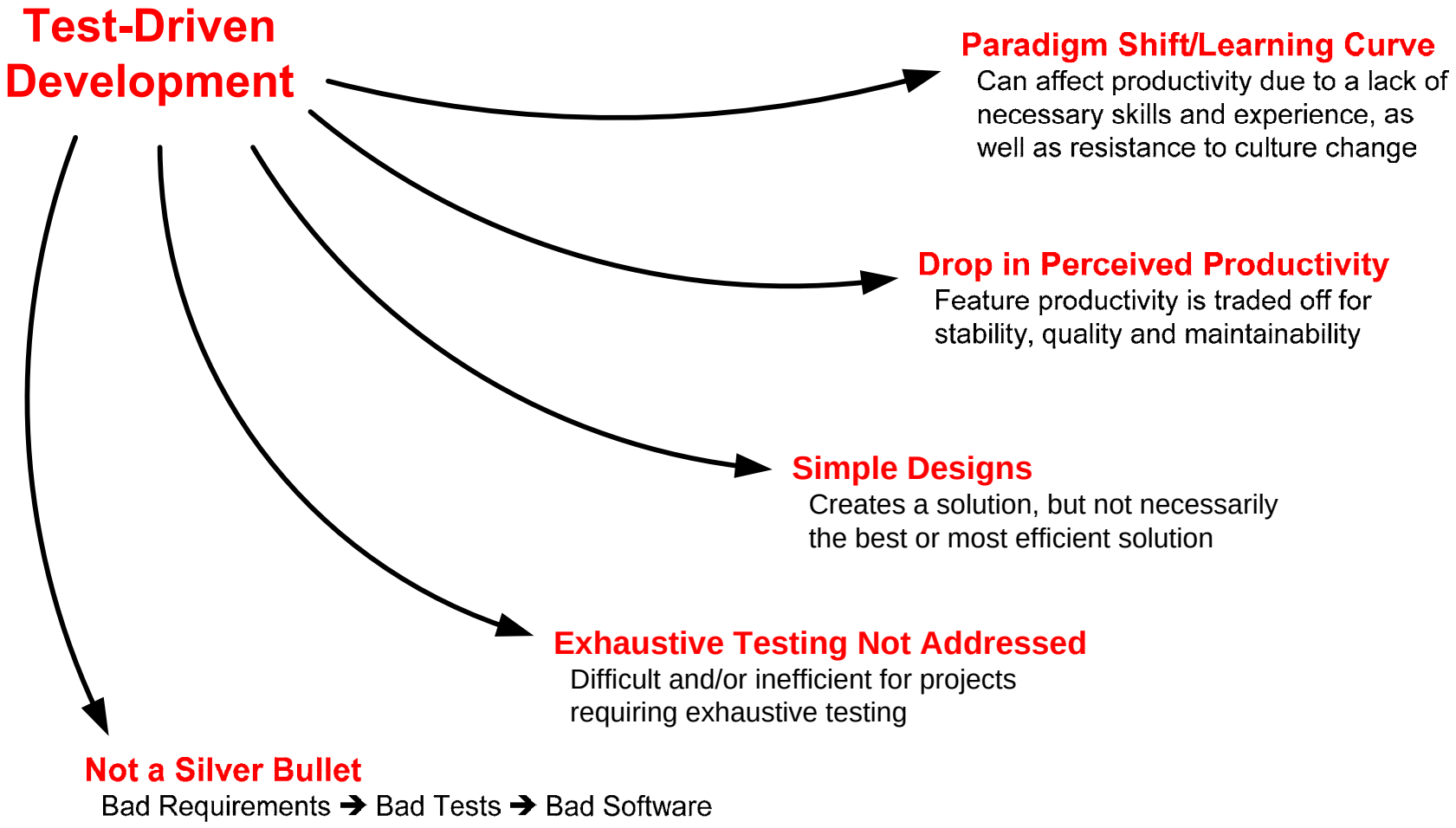
From Details to Done

- Develop acceptance test scenarios from groups of related features.
- Develop integration tests for components of a simple, initial design.
- Develop unit tests and components using the red-green-refactor approach and mock objects.
- Integrate components by replacing mock objects with actual components and executing unit and integration tests.
- Execute acceptance test scenarios to ensure all functionality is complete.

Benefits of Test-Driven Development



Drawbacks of Test-Driven Development



End

Very Simple TDD Example – Hello World

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
   }
}
```

Compilation Error

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
   }
}
```

Greeter

```
public class Greeter
{
   Greeter()
}
```

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), "Hello World")
   }
}
```

Compilation Error

Greeter

```
public class Greeter
{
   Greeter()
}
```

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), "Hello World")
   }
}
```

Test Failure

Greeter

```
public class Greeter
{
   Greeter()

   String getGreeting()
   {
      return ""
   }
}
```

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), "Hello World")
   }
}
```

Greeter

```
public class Greeter
{
   Greeter()

   String getGreeting()
   {
      return "Hello World"
   }
}
```

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), "Hello World")
   }
}
```

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   Greeter()

   String getGreeting()
   {
      return greeting
   }
}
```

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   const String expectedGreeting = "Hello World"

   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), expectedGreeting)
   }
}
```

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   Greeter()

   String getGreeting()
   {
      return greeting
   }
}
```

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   const String expectedGreeting = "Hello World"

   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), expectedGreeting)
   }
}
```

Compilation Error

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   private Greeter()

   static Greeter GetInstance()
   {
      return new Greeter()
   }

   String getGreeting()
   {
      return greeting
   }
}
```

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   const String expectedGreeting = "Hello World"

   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = Greeter.GetInstance()
      Assert(myGreeter.getGreeting(), expectedGreeting)
   }
}
```

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   private Greeter()

   static Greeter GetInstance()
   {
      return new Greeter()
   }

   String getGreeting()
   {
      return greeting
   }
}
```

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   const String expectedGreeting = "Hello World"

   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Assert(Greeter.GetInstance().getGreeeting,
             expectedGreeting)
   }
}
```

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   private Greeter()

   static Greeter GetInstance()
   {
      return new Greeter()
   }

   String getGreeting()
   {
      return greeting
   }
}
```